TensorFlow Implementation for SNAIL and RL^2

Zeqi Li Lan Xiao Zihang Fu
Department of Computer Science
University of Toronto
Toronto, ON M5S 2E4
{lizeqi, lxiao, fuzihang}@cs.toronto.edu

Abstract

Deep reinforcement learning excels in problems with lots of training data available, but behaves less superior when the data is scarce. Also, the learned model tends to overfit to its trained task, which makes it difficult to generalize to similar tasks and transfer prior knowledge to new domains. The Simple Neural AttentIve Learner (SNAIL) and RL^2 are two novel meta-learners proposed to bridge this gap. In this project, we will present a TensorFlow implementation of SNAIL and RL^2. We will evaluate both meta-learning algorithms on several heavily-benchmarked tasks including multi-armed bandits, tabular Markov Decision Processes (MDPs), and visual navigation.

1 Introduction

Recent advancement in reinforcement learning (RL) has allowed well established techniques to be applied to many complex and large-scale task domains such as Go [1] and visual navigation [2]. Many key advances were made possible as a result of the development of the stable integration of RL with non-linear function approximation using deep neural networks. These learning algorithms, however, lack the ability to adapt quickly to changing tasks with scarce training examples. In contrast, human learners are able to achieve reasonable performance on a wide range of tasks with comparatively little experience.

Meta-learning seeks to bridge this gap by training the learner on a distribution of similar tasks and effectively reuse past experience on new tasks. We refer to such learners as meta-learners. The key idea underlying many meta-learners is the ability to integrate its prior knowledge with a small amount of new information without overfitting to new data. The form of prior knowledge and new data depends on the tasks. At an structural level, it is useful to conceptualize meta-learning as consisting two learning systems: one fast meta-learner which adapts to each new task from a few training examples, and a slower learner that works across the distribution of similar tasks to fine tune and improve the meta-learner.

In this work, we focus on the Simple Neural AttentIve Learner (SNAIL) [5] and Fast Reinforcement Learning via Slow Reinforcement Learning (RL^2) [3], two recently proposed meta-learning frameworks. They try to use different techniques to solve a specific problem in meta-learning, which is meta-learner's inability to internalize and refer to past experience. In particular, SNAIL combines temporal convolutions with causal attention while RL^2 uses recurrent neural networks (RNNs) to tackle this problem. We implement these two existing methods, in the hope of building a more concrete understanding of their impressive performance and potential limitations. We evaluate both algorithms on several heavily bench-benchmarked meta-RL tasks proposed in the original paper, including multi-armed bandits,tabular MDPs and visual navigation. We realize the algorithms in TensorFlow and assess the performance based on the experiments proposed in the original papers.

2 Method

2.1 PPO

As we use the PPO [11] implementation released by OpenAI Spinning Up, we refer the reader to the original release for implementation details [1]. In this section, only important changes that we make to the original code will be discussed.

Besides the range clipping mechanism proposed in the original PPO paper, their implementation also introduces a KL-divergence constraint, which is used as the early stopping criteria in the optimization loop when the KL-divergence exceeds the target threshold. In our experiments, we relax this constraint as we find out it slows down the learning of the agent significantly by frequently cutting off the π updating loop after the first iteration. This also results in a huge sample inefficiency. We also try changing the objective function by adding an entropy term to it, which is suggested in the PPO paper but is not implemented in their code. However, we find this prone to numerical errors and abandon this change in our final version of PPO.
Table 1 shows the hyperparameters we use for PPO. These hyperparameters are kept constant for all experiments.

Table 1: Hyperparameters for PPO

Clip ratio	0.2
Learning rate	3e-4
GAE λ	0.97
Discount factor	0.99
Policy training iteration per PPO update	100

2.2 RL^2: Fast Reinforcement Learning Via Slow Reinforcement Learning

RL^2 is proposed by Duan et al in their paper [3]. In their paper, they propose a meta-RL agent with a recurrent layer in its policy network. The agent is trained on similar but different tasks sampled from a task distribution, with certain episodes of repetition on each task. At each time step, the current environment state x, the last action a, and the last reward r, are concatenated altogether as the input to the recurrent layer with Gated Recurrent Units (GRUs). The recurrent layer is then connected to a fully connected layer which is responsible for outputting the probability distribution of actions. The recurrent layer is used to memorize the past experience of the current task. Therefore, the weights connecting the recurrent layer and the output layer define a exploration and exploitation policy as a function of the agent's past experience with the task. As argued in the paper, this architecture enables the agent to learn a general policy for the entire distribution of tasks, as opposed to specializing in a specific sampled task.

Implementation Details Most of our implementation details follow strictly from those proposed in the original paper. In this project, the recurrent layer contains 256 orthogonally initialized GRUs, which are then fully connected to the output layer. The weights in the output layer are initialized with Xavier initializer. All biases are initialized with zero. The input to the network is a 1-D vector concatenation of the state, action, reward, and termination flag. We use TensorFlow's `dynamic_rnn` function to build the recurrent layer, as it allows batch input with variable sequence length in training and can automatically reset GRUs' hidden states at the end of a sequence. Each sequence is defined as a series of input vectors during the same task, and the GRUs' hidden states are carried over from one time step in the sequence to the next until the end of the sequence.

For each iteration, we first run the model on n tasks, with m episodes per task and a maximum of p steps per episode. As an episode could terminate before maximum number of steps p is reached, each episode will have variable time steps. In order to ensure all sequences have the same length, we pad zero vectors to the sequence if an episode terminates before p is reached. PPO objective function is optimized using the Adam Optimizer with default hyperparameters [4].

We implement weight normalization without data-dependent initialization to improve the optimizability of RL^2. Specifically, we implement weight normalization for both the RNN model and fully

[1] https://spinningup.openai.com/en/latest/_modules/spinup/algos/ppo/ppo.html#ppo

connected layer. The implementation for RNN is a drop-in replacement for the `GRUCell` [2] class, which is the main architecture used in RL^2. There are two sets of weight in a GRU cell, namely the weights for the inputs and the weights for the RNN states. Both sets of the weights are reparameterized using orthogonal initializer. For the fully connected layer, the reparameterized weights use Xavier initializer and the scalar controlling the magnitude uses zero initializer.

2.3 A Simple Neural Attentive Meta-Learner

SNAIL is universally applicable to domains in both supervised and reinforcement learning [5]. In this work, we focus on the reinforcement learning setting. Mishra et al. point out traditional RNN architectures suffer from a temporally-linear dependency which bottlenecks their capacity to perform sophisticated computation on a stream of inputs. Instead, SNAIL performs dilated 1D-convolutions over the temporal dimension. These temporal convolution (TC) are causal to ensure future timestep does not affect past outputs [6]. Although TC provide higher-bandwidth access to past information, it has coarser access to inputs that are further back in time. To overcome this limitation, SNAIL uses soft attention [7] to pinpoint a specific piece of information from all its past experience gathered by TC layers. The use of attention enables SNAIL to learn what pieces of information to pick out from its experience and how to effectively represent the features.

Our implementation strictly follows the setup in the original paper. In practice, SNAIL is trained over batches of tasks. Therefore, the shape of input to the network is [batch_size $\times$ sequence_length $\times$ input_dimensionality]. For the dilated convolution, we use the 1D convolution layer supported by keras. One thing that we found particularly challenging is the implementation of causal mask in TensorFlow. Unlike PyTorch, which uses dynamic computational graph and supports broadcastable masking operation natively, masking operation for dynamically changing tensor in TensorFlow is highly non-trivial. The way we implemented it is to pre-construct a masking template based on the number of episodes and the horizon for each episode number of episodes and the horizon for each episode and replicate the template dynamically depending on the dimension of the input, which is different between inference and learning.

3 Related Work

The concept of Meta-RL has been previously discussed by Schmidhuber et al., who firstly combines meta-learning with reinforcement learning [8]. More recently, researchers have been focusing on using neural networks to improve the performance of meta-learners. Several neural network based algorithms have been proposed, including LSTM A2C, RL2, and SNAIL [3][5][9].

This project is strongly related to works done by Duan et al.[3] and Mishra et al. [5]. Duan et al. [3] use Gated Recurrent Units (GRUs) as the meta-learning agent. Mishra et al. propose SNAIL, which combines temporal convolutions and soft attention in its architecture [5] as the meta-learner. Both models achieve comparable performance in various task domains.

Both RL2 and SNAIL use first-order implementation of Trust Region Policy Optimization (TRPO) [10] to optimize the policy network. In this project, we use Proximal Policy Optimization (PPO) with clipped surrogate objective as an alternative, for its simplicity in implementation and its computational efficiency in model training [11]. In this project, we borrow the code of PPO TensorFlow implementation released by OpenAI Spinning Up and make changes to suit our needs.

The implementation of SNAIL in PyTorch for few-shot image classification is available online [3]. However, our implementation and analysis of SNAIL and RL^2 will be done in TensorFlow, which has a larger community and supports more production-ready deployment options [12]. Experiments in visual navigation are conducted using the VizDoom environment [13]. In experiments with multi-armed bandits and tabular MDP problems, we follow the problem setup from Duan et al. and use results from Gittins index and OPSRL as theoretical optimums respectively [3][14]. The implementation for both multi-armed bandit and tabular MDP environments are borrowed from an open source repository with some changes [4].

[2] https://www.tensorflow.org/api_docs/python/tf/nn/rnn_cell/GRUCell

[3] https://github.com/eambutu/snail-pytorch

[4] https://github.com/tristandeleu/pytorch-maml-rl/tree/master/maml_rl/envs

To increase the capability and accelerate the training of RL^2, weight normalization without data-dependent initialization is applied to all weight matrices as suggested by the RL^2 paper. It effectively normalizes the weights in a neural network to unit length and introduces a separate scalar to control the magnitude of the weights. The resulting optimization problem with the new parameters is shown to be better conditioned than the original problem [15]. The key insight behind their technique is to treat the direction and magnitude of the weights as separate variables. This allows one to rotate the weights without affecting their respective magnitudes and vice versa. In this work, we implement weight normalization for GRU cell. Fully connected layer is natively supported by TensorFlow.

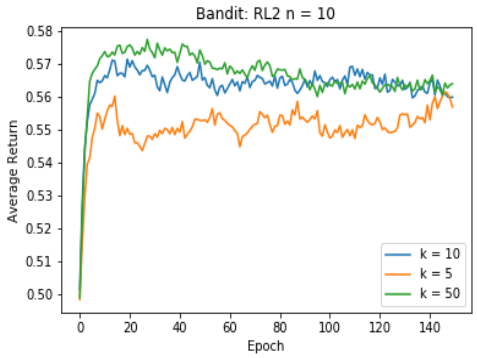
4 Experiment

4.1 Multi-Armed Bandit

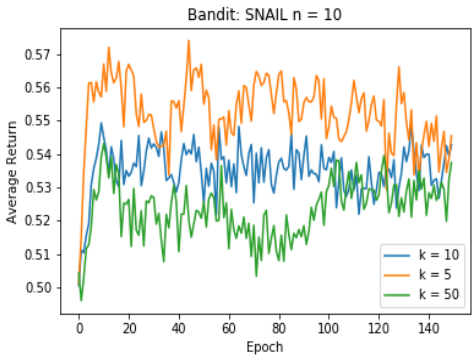
We evaluate both RL^2 and SNAIL on this task to directly compare performance of both models and the results of our implementation with proposed results. We run the experiments with N = {10, 100, 500} and K = {5, 10, 50}, with k denoting the number of arms and N denoting number of episodes the agent is allowed to interact within a specific task. In this case, each time step is a complete episode as the environment is stateless. Therefore, the state and the termination flag are trivially set.

For each epoch of training, we use 250,000 sampled interactions for RL^2; due to the complexity of the SNAIL model and our limitation on computation resource, we are not able to perform the experiment with the same batch size for SNAIL. For SNAIL, we use 25,000 sampled interactions per epoch of training.

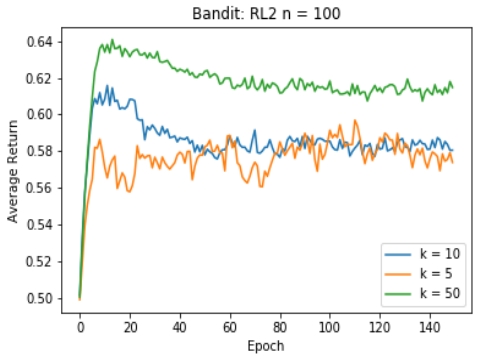
Figure 1 shows the results of multi-armed bandit experiment on RL^2 and SNAIL, respectively. See Appendix for more detailed experimental results.



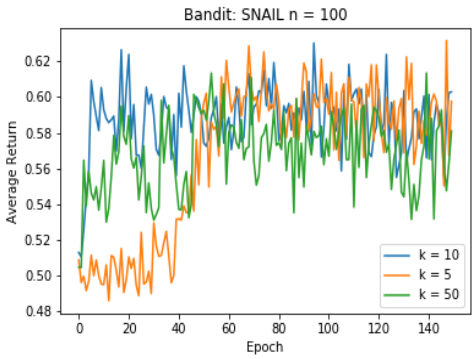
(a) Performance of RL^2 at n = 10



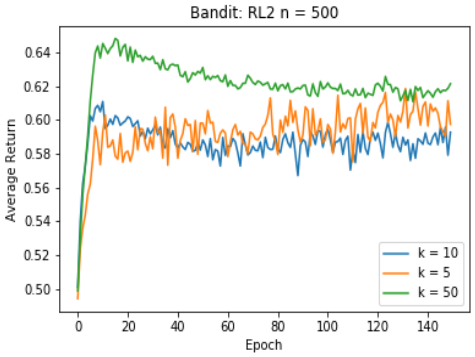
(b) Performance of SNAIL at n = 10



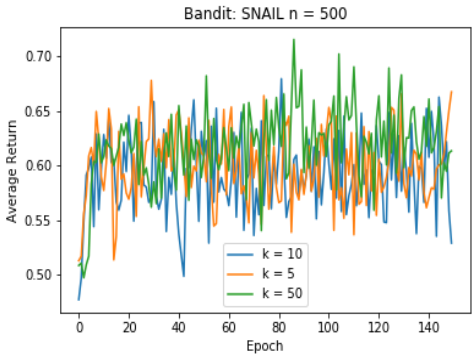
(c) Performance of RL^2 at n = 100



(d) Performance of SNAIL at n = 100



(e) Performance of rl^2 at n = 500



(f) Performance of SNAIL at n = 500

Figure 1: Learning curves for RL^2 and SNAIL on multi-armed bandit (cont.)

Table 2: Best results achieved with RL^2 and SNAIL on multi-armed bandit problems

Setting	RL^2	Paper	SNAIL	Paper
n=10, k=5	0.56	0.67	0.58	0.66
n=10, k=10	0.57	0.67	0.55	0.67
n=10, k=50	0.58	0.68	0.54	0.67
n=100, k=5	0.58	0.79	0.63	0.79
n=100, k=10	0.62	0.83	0.63	0.83
n=100, k=50	0.64	0.85	0.62	0.85
n=500, k=5	0.60	0.80	0.68	0.82
n=500, k=10	0.61	0.86	0.67	0.86
n=500, k=50	0.65	0.88	0.73	0.88

The performance of RL^2 in all experiment settings reaches its maximum in around 20 epochs, and starts to drop after that. Inspecting the behavior of the agent, we find that after around 20 epoch of training the agent starts to become fixated on 2 to 3 arms and barely explore the rest. We believe this might be related to relaxing the KL-divergence constraint in the PPO implementation, which makes KL-divergence increase much faster. This results in unexpected large update in a single step, which is quite hard to unlearn later on. However, if the KL-divergence is kept, the learning is so slow that the agent fails to learn anything useful in reasonable time. This suggests a possible future work in balancing the learning speed and KL-divergence.

We observe the trend that average return increases as n becomes larger for both RL^2 and SNAIL. This is expected because larger n gives model more chance to interact with the environment to adjust its internal state to the true distribution of the task and exploit the more compromising arms. For k, however, we only observe this effect in RL^2. We believe this is also a result of its smaller batch size, causing the unstableness in the output. We also observe that there is a gap between the performance of our implementation of both RL^2 and SNAIL and their official performance as in their paper. We believe the KL-divergence issue discussed earlier is a major cause for this performance gap. Furthermore, for SNAIL, we believe another reason for this gap is that we use a smaller batch size in training.

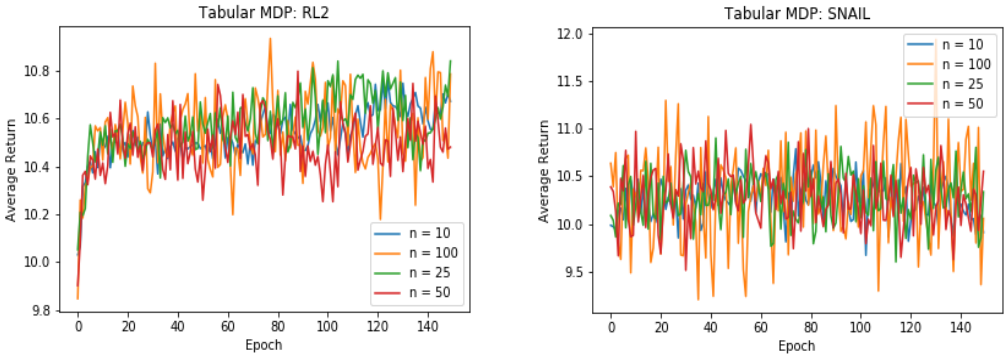
4.2 Tabular MDPs

Similarly, we evaluate the performance of both models on tabular MDPs, also following Mishra et al [5]. Each MDP had 10 states and 5 discrete actions. The rewards follow a normal distribution with unit variance where the means are sampled from $\mathcal{N}(1,1)$. The transitions are sampled from a flat Dirichlet distribution. Unlike the bandit problems, tabular MDP involves sequential decision making and better characterizes the challenges in solving MDPs. We allow each agent to interact with a MDP for N episodes of length 10. The input to the agent includes the current state, the previous action,

the previous reward, and a termination flag. We train RL^2 with batch size = 250,000. As SNAIL is more computationally demanding and time consuming, batch size is reduced to 25,000 matching with multi-armed bandit settings.

Table 3: Best results achieved with RL^2 and SNAIL on tabular MDPs

Setting	RL^2	SNAIL	Paper
n=10	10.76	10.79	13.29
n=25	10.84	10.86	14.14
n=50	10.80	11.00	13.85
n=100	10.88	10.98	12.28



(a) Performance of RL^2

(b) Performance of SNAIL

Figure 2: Learning curves for RL^2 and SNAIL on tabular MDPs

The results when $N = \{10, 25, 50, 100\}$ are displayed in figure 2. It can be observed from figure 2b that despite of the high variance, SNAIL generally achieves the same level of performance with much fewer iterations than RL^2 (figure 2a). This behaviour matches with the original paper. We also observed that SNAIL exhibits higher variance while learning. This can be explained by the use of smaller batch size due to limited computational power. Due to time constraints, we are only able to run tabular MDPs for 150 epochs before reporting the results. In both papers' setup, 1,500 epochs are trained when $N = 10$, and 6,000 epochs are trained for even larger Ns. This large gap in the number of training epochs between our results and the papers' results make them not directly comparable. Instead, we read the corresponding rewards from the original paper' plots and compare those with ours. For both models, our meta-learners achieve similar performance when N= 100, but perform less satisfying for smaller Ns. As we previously discussed in section 4.1, we suspect the KL-divergence term in PPO also has a large impact on the performance. Moreover, SNAIL does not specify their initialization method, and we use the default initializer in TensorFlow. Using different initializers such as Xavier initializer could potentially improve our performance. Last but not least, we observe that adding weight normalization does not boost our performance by the amount suggest in the weight norm paper. Instead, adding weight normalization sometimes cause our model to attain higher variance. Hence, we believe our weight normalization implementation can be refined to further improve our performance.

4.3 Visual Navigation

As mentioned in our project proposal, we plan to evaluate both algorithms on 3D visual navigation tasks on VizDoom environment. However, as this task requires a few additional convolutional layers to process image data, the size of the model increases and training slows down dramatically (more than 1 million parameters for both pi and v network). We test RL^2 on the basic VizDoom environment

[5] with affordable batch size (10,000, instead of 50,000 as in the paper) but observe no signs of learning. In addition, we find that we are unable to reproduce the exact visual navigation environment settings used by the RL^2 and SNAIL paper in a timely manner (specifically, the paper requires a 'hitting-wall' penalty in the navigation environment, which is not directly supported by VizDoom's configuration file and can only be realized by writing new scripts to detect whether an agent touches the wall). Due to above difficulties, our time constraints and computation resource for this project, we decide to evaluate the algorithms on tabular MDPs extensively instead.

5 Future Works

As mentioned in section 4.1, one of the possible future work is to further research what causes the performance drop in RL^2 after about 20 epochs. This includes further studying the necessity of the KL-divergence constraint in the PPO implementation.

Another future work is to train SNAIL on multi-armed bandit tasks with the same batch size as RL^2 on some more powerful computation servers. Other experiments, such as visual navigation, could be conducted to do more direct comparison between two algorithms. Another interesting idea would be to train our meta-learners, both RL^2 and SNAIL, more on the tabular MDPs. Due to the lack of training, we do not observe the impressive performance as proposed in the original papers, but given more training it is likely that we can unfold more satisfying results. We also want to refine our implementation of weight normalization to potentially improve our performance because, this reparameterization technique can be generally useful in training large scale network.

Acknowledgments

We would like to thank Jimmy Ba, Tingwu Wang, Michael Zhang, Bryan Chan and the OpenAI team for useful discussion and feedback.

References

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

[2] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuo- motor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[3] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, RL2: Fast Reinforcement Learning via Slow Reinforcement Learning, ArXiv e-prints arXiv:1611.02779, 2016.

[4] D. Kingma and J. Ba. Adam: A method for stochastic opti- mization. *ICLR*, 2015.

[5] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. International Conference on Learning Representations *ICLR*, 2018.

[6] Aaron van den Oord, Sander Dieleman, Heig Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016a.

[7] Ashish Vaswani, Noah Shazeer, Jakob Uszkoreit, Llion Jones, Aidan Gomez N., Lukas Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv* :1706.03762, 2017a.

[8] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-... hook.) *Diploma thesis*, Institut f. Informatik, Tech. Univ. Munich, 1987

[9] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. (2016) Learning to reinforcement learn *arXiv preprint arXiv* :1611.05763

[10] Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy op- timization. In International Conference on Machine Learning *ICML*, 2015a.

[5] In basic VizDoom environment, the agent is free to move horizontally and fire. The goal is to kill the monster appearing on the other side of the screen

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017

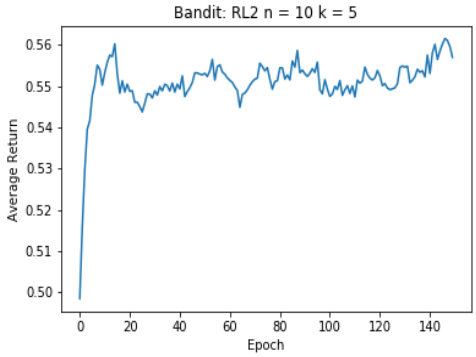
[12] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv*:1603.04467

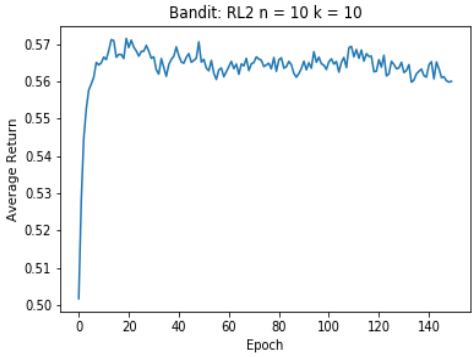
[13] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. (2016) Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv*:1605.02097

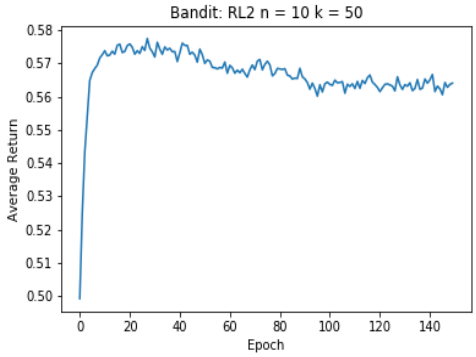
[14] John C Gittins. (1979) Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society*. Series B (Methodological), pp. 148–177, 1979.

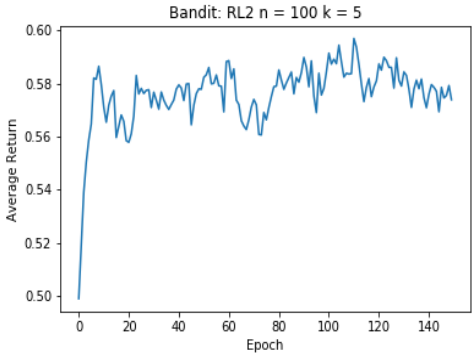
[15] Tim Salimans and Diederik Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NIPS*, 2016

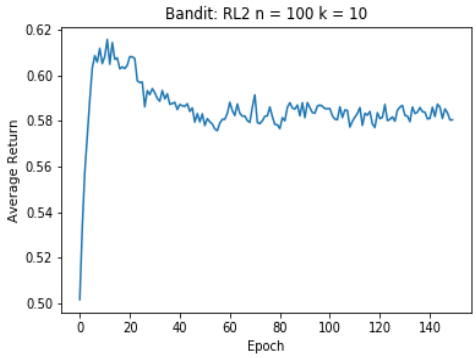
Appendix: Results for multi-armed bandit

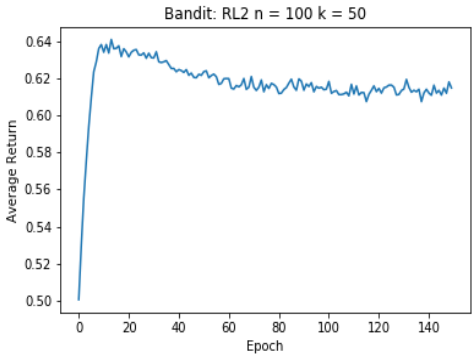


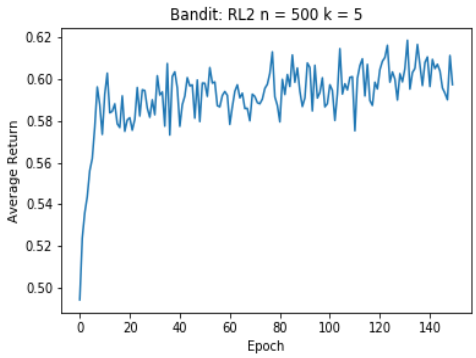


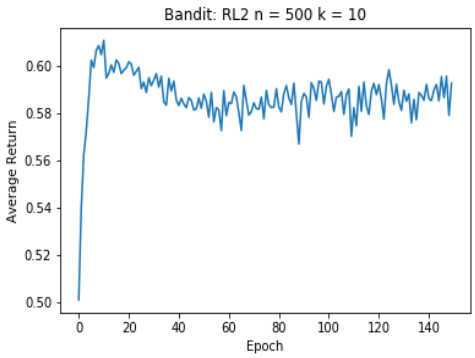












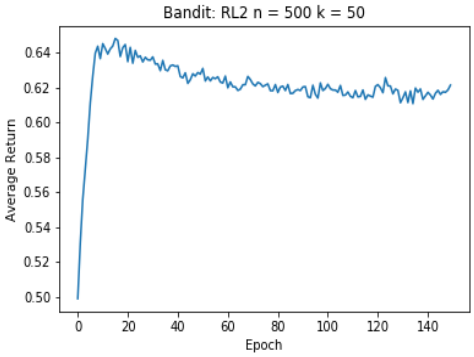
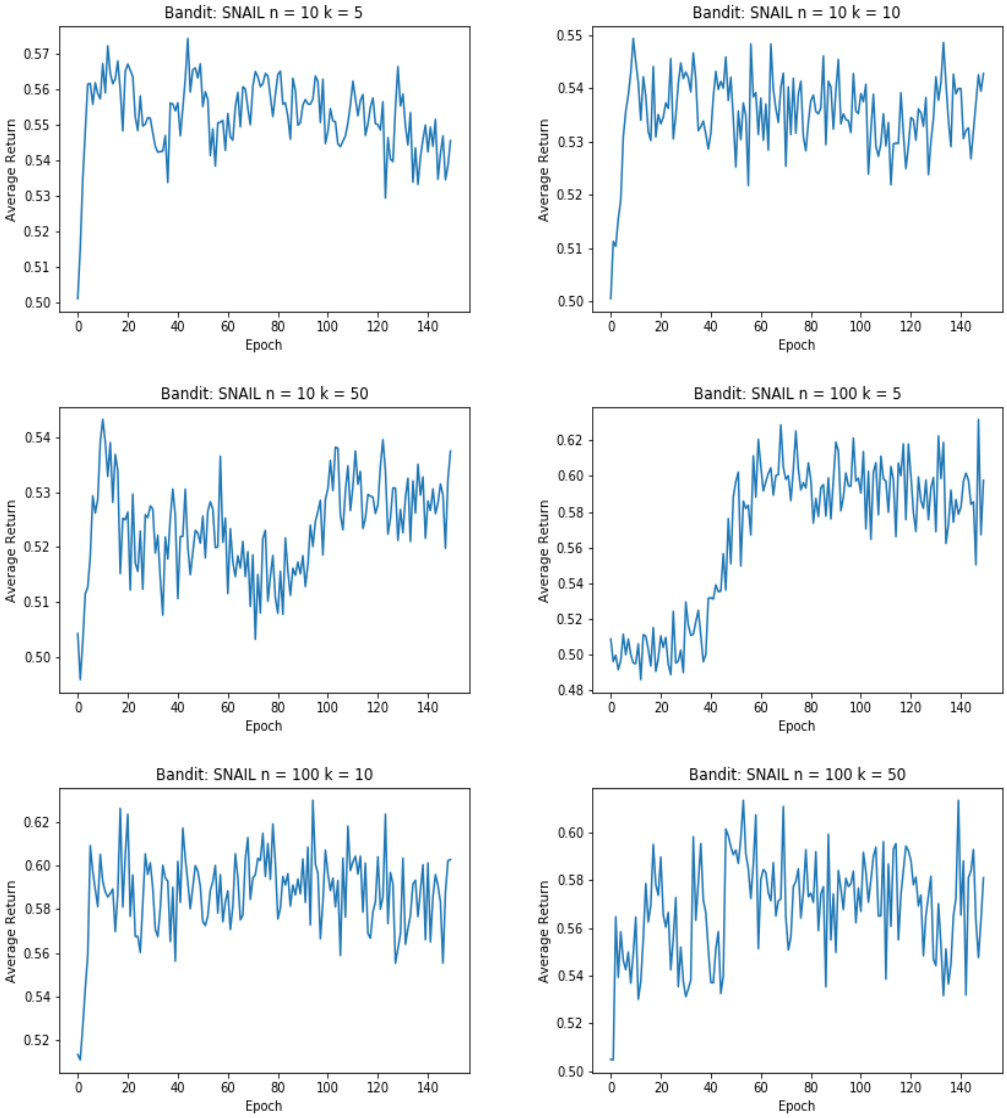


Figure 3: Learning curves for RL^2 on multi-armed bandit



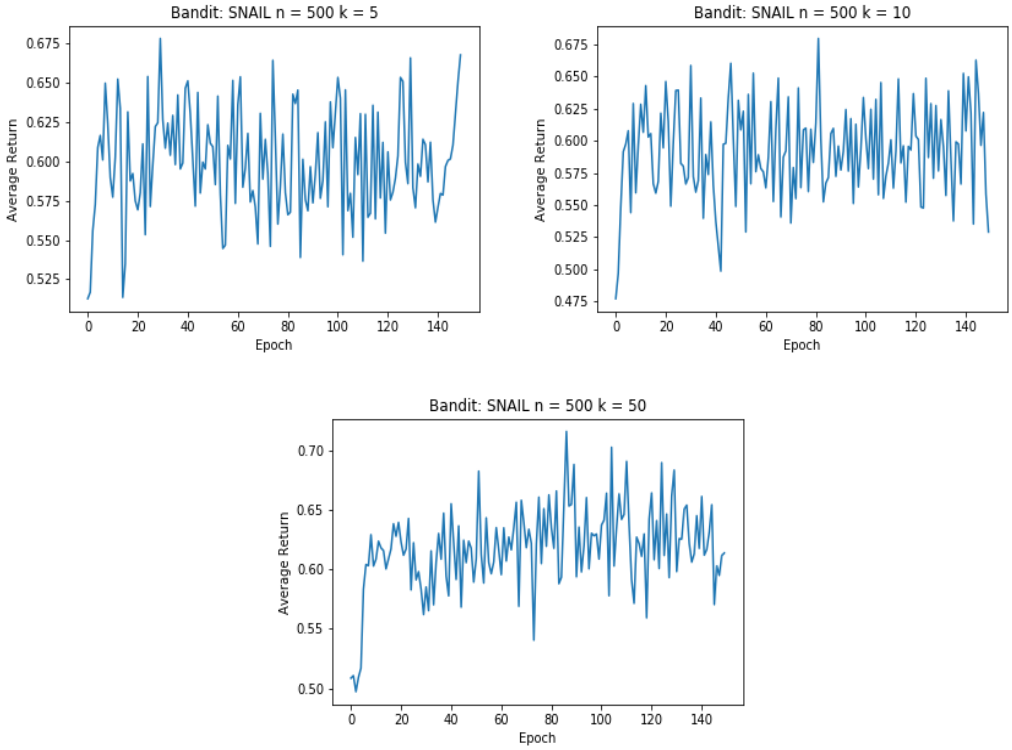


Figure 4: Learning curves for SNAIL on multi-armed bandit